M. Tech. (Computer Science) Dissertation Series

DEVELOPMENT OF A SOFTWARE FOR TWO DIMENSIONAL PARAMETRIC-SHAPE GENERATION AND MANIPULATION

a dissertation submitted in partial fulfillment of the requirements for the M. Tech. (computer Science) degree of the Indian Statistical Institute

By

Banikumar Maiti

Under the supervision of

Dr. Probal Sengupta

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Probal Sengupta (System Analyst, Computer Vision & Pattern Recognition Unit) for providing me invaluable support throughout this project. During the course of this project he has always given me helpful suggestions whenever I was in trouble.

I would also like to thank Mr. K. Rajesh Babu , my classmate, for his numerous valuable suggestions regarding Object-Oriented Design. Finally, I would like to thank all my classmates for their moral support throughout this project.

Banikumar Maiti

Certificate Of Approval

*This is to certify that the dissertation work entitled "**Development of a Software for Two-dimensional Parametric Shape Generation and Manipulation**", submitted by Banikumar Maiti, in partial fulfillment of the requirements for M. Tech in Computer Science degree of the Indian Statistical Institute is an acceptable work for the award of the degree.*

Date : 31/7/98

[illegible]
(Supervisor)

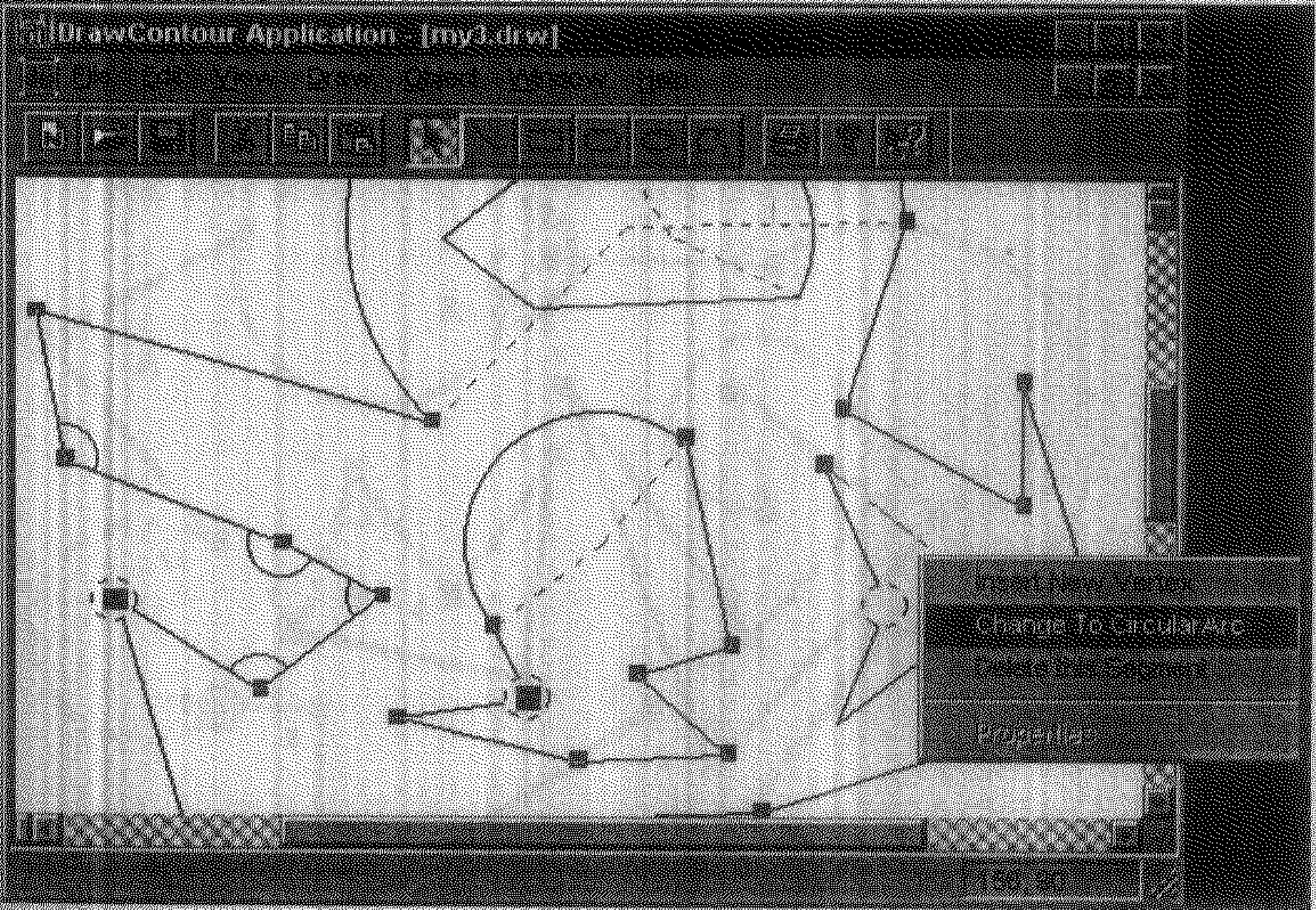
S. Chattopadhyay
(External Examiner)

Contents

ABSTRACT

The aim of this project is to develop a Software (named as **"DrawContour"** *) for easy generation and manipulation of 2-dimensional "Shapes". This manipulation of Shapes is a simulation of cutting out complex shapes from metal-sheets, before they are actually done. The beauty of the software is that it can store the whole shape in a archive of very small space, compared to bitmap storing. It can also export the data of the shape (however complex it might be) in a convenient text format. The Software is also supposed to parse a "Shape" presented to it, in a specific text format, and can draw the "Shape" accordingly.*

It Works ....


DrawContour Application - [my3.drw]
File Edit View Draw Object Window Help
Insert New Vertex
Change To CircularArc
Delete this Segment
Properties

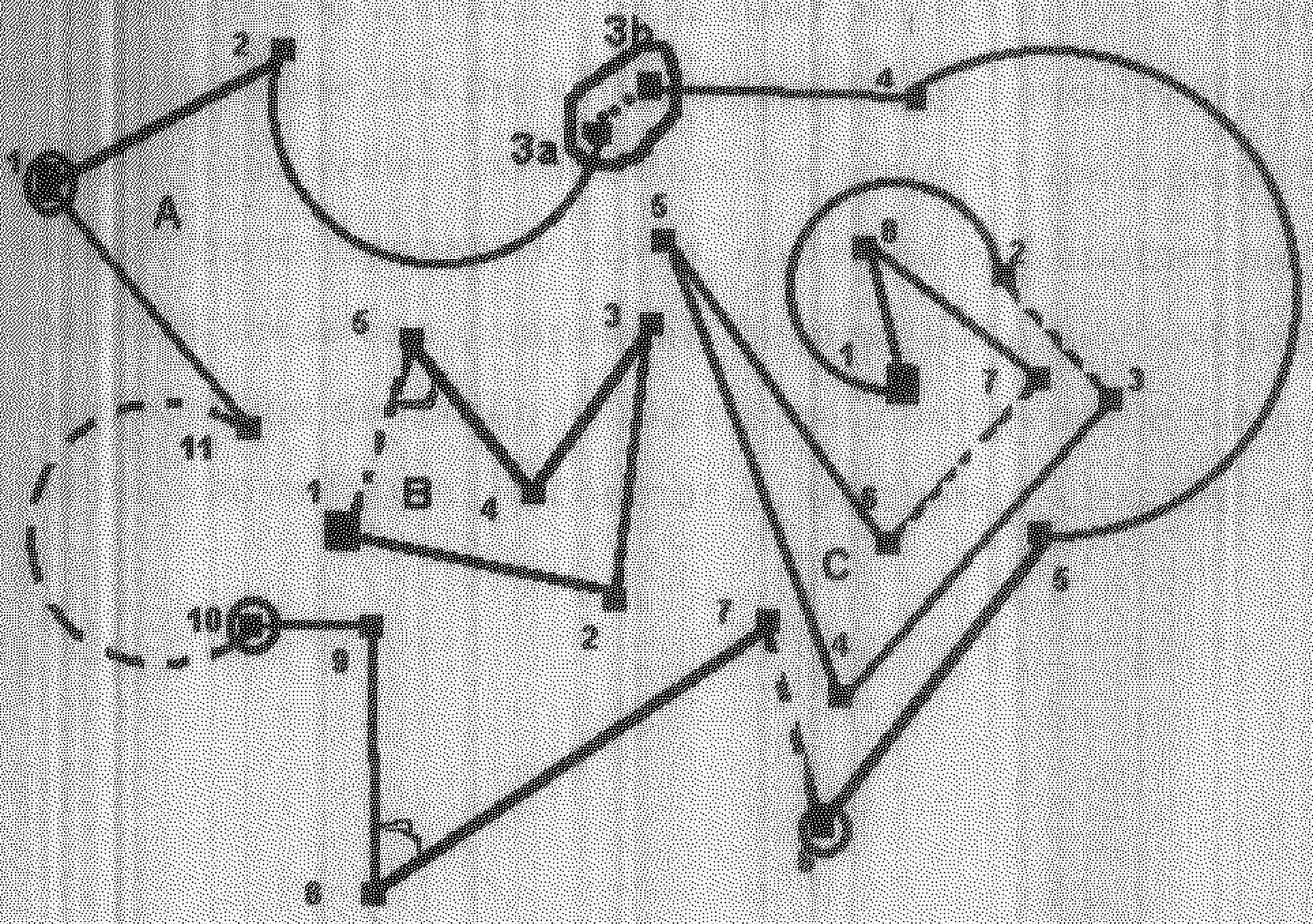
"DrawContour" Application Running....

Introduction :

A polygon whose every side (called an *'edge'*) is a particular curve between two end-vertices, is called a *'generalized polygon'*.

A generalized polygon that is closed and of which no edges intersect each other, is called a *'Contour'*.

A figure with only ONE *outer contour* and zero or more *inner contours* , where no inner contour overlaps with each other or with the outer contour, is called a *'Shape'*. The diagram below, explains one such possible shape.



The above shape consists of 1 outer contour (which is always the case), viz. A, and 2 inner contours, viz. B and C. The dotted edges represent edges of fixed lengths. The angles marked, represent constant angles. Vertices those are encircled are fixed or non-movable vertices. The edges which are arcs, have got their own centers, radii and arc-lengths. Besides, an arc-edge may also be of fixed length. We will say that, e.g., edge 5-1 makes a fixed angle with previous edge 4-5, in contour B, in vector sense; i.e. the concerned fixed angle is measured from ray 4-5 towards ray 5-1. The vertices are subject to movements, as clicked-n-dragged by the user, ofcourse obeying all the interrelationships provided by him/her.

Vertex no. 1 in each of the contours, is called the *Origin* of the corresponding contour. These are assumed to be fixed vertices. This special vertex is the starting point in course of drawing a particular contour.

There is also a concept of another particular type of vertex, called *Joint Vertex*. These are the vertices formed dynamically if need arises to break-up a fixed vertex, to maintain the sanity of the figure. Each such vertex has two component-vertices, one fixed and another movable. The link between these two components behave like a rubber-band, and in some way measures the deviation of the current configuration from the actual one. In the above figure, in contour A, we show such a vertex 3a-3b.

Model Description :

The project work deals with the development of a **Software**, which can let the user create, add constraints on the constituting segments and contours, and subsequently **manipulate** a "*Shape*". The constituting segments of a contour may be either a line segment or a circular arc.

When the user creates a new document, he has to start with a *Skeleton Contour*, which is either a square or a triangle. Later on he/she may go on :

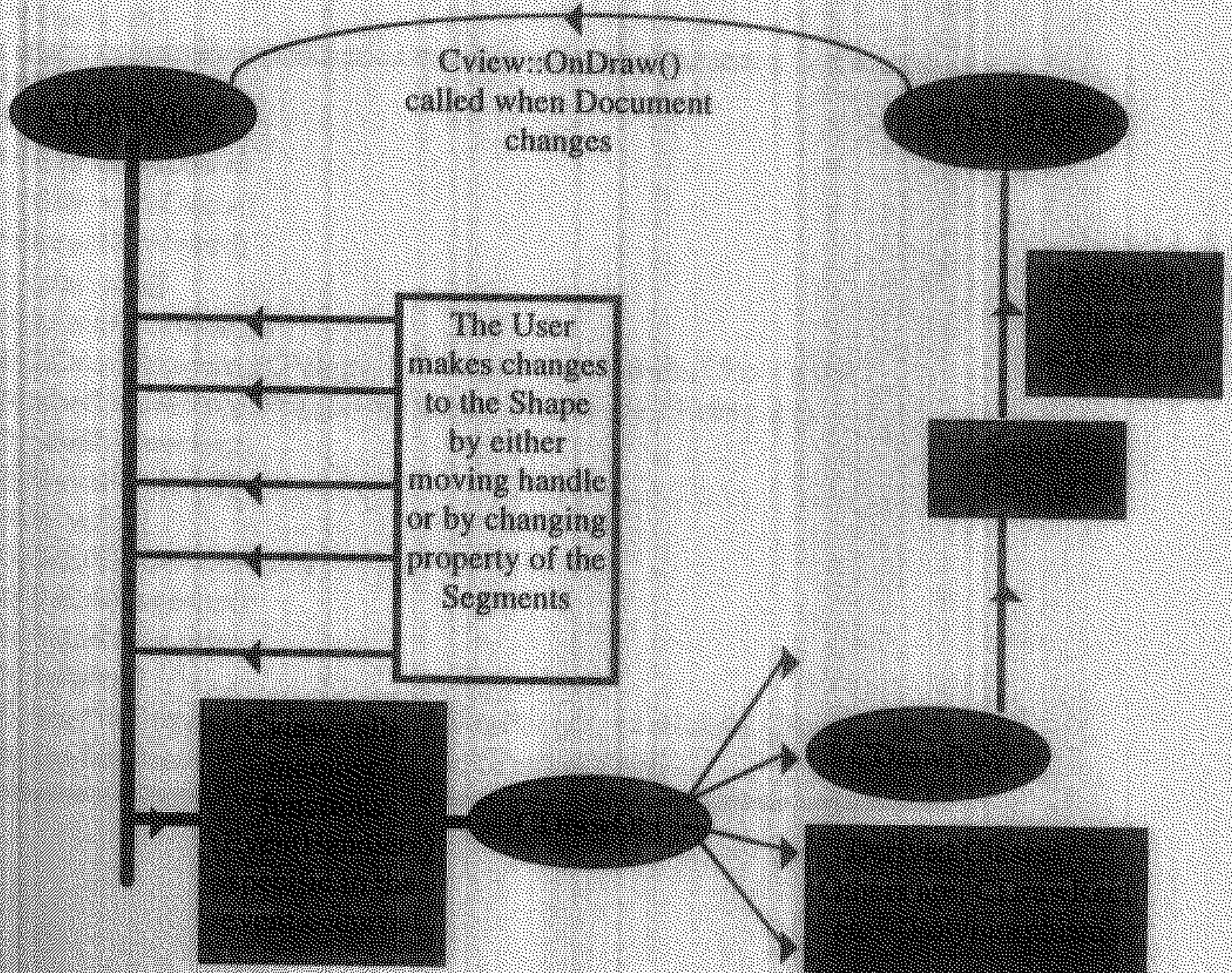
- Inserting more vertices between two end vertices.
- Changing a line segment to a circular arc, assuming the end vertices to be fixed for the time being.
- Specifying restriction (i.e. fixed/variable) on the segment lengths.
- Specifying relations among the adjacent segments, e.g. fixing the vector angle of a line with the previous line.
- Specifying restriction, in case of arcs, on the radius.
- Fixing some of the vertices.
- And finally, drag a vertex("*handle*") to an arbitrary point on the drawing paper, to see the possible new shape. In case there doesn't exist any possible shape with the current relationships, the user is informed about the failure and with possible suggestions.

Method Used :

For developing the software, which is supposed to run on Microsoft Windows 95 or NT platform, we used Document/View Architecture of Windows (Event-driven) Programming. I opted for Object Oriented programming paradigm. This was needed so as to manage the whole software efficiently, and, as real-life problems best can be described by Objects & interactions between them.

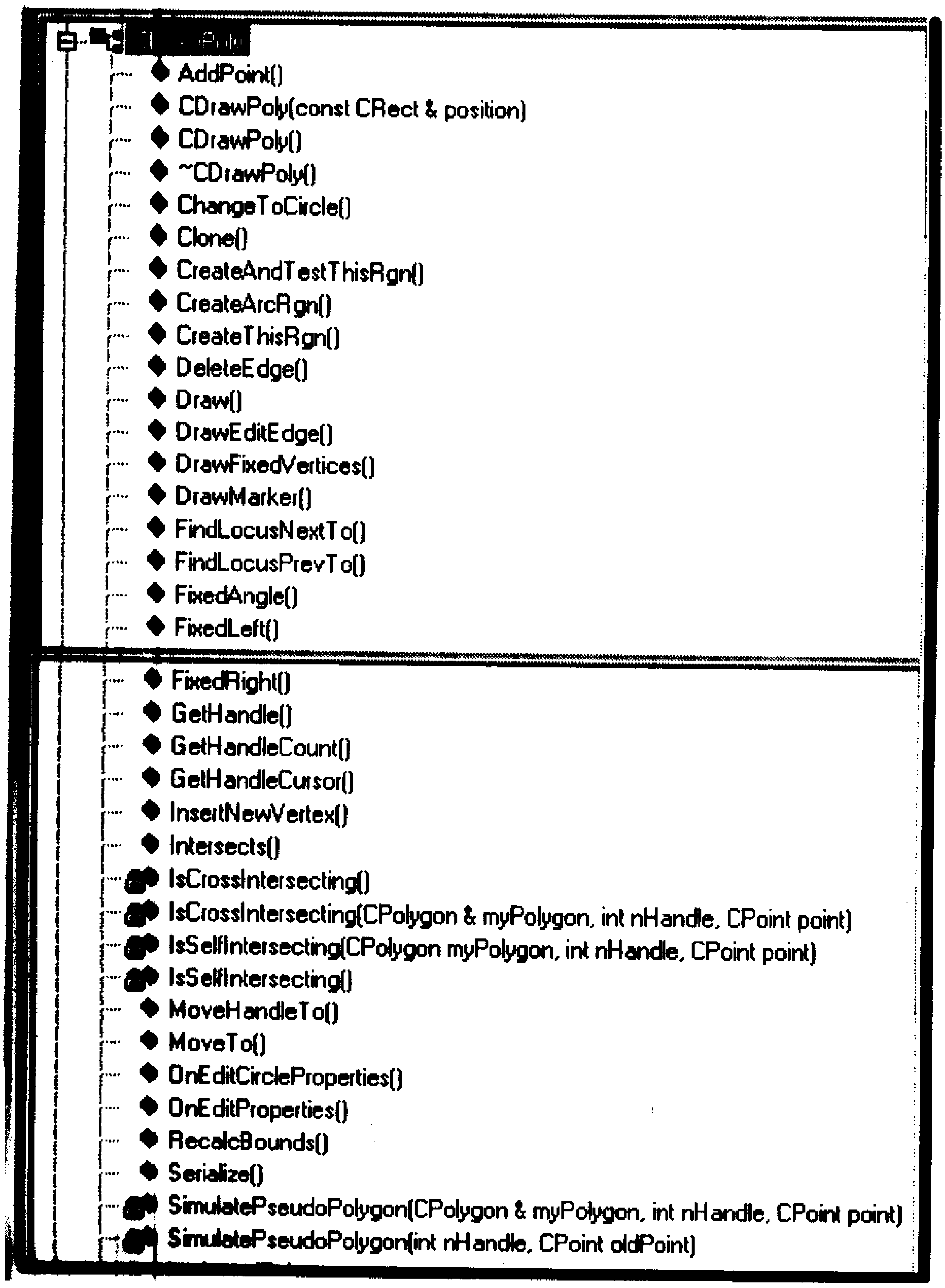
So, the initial job was to prepare a (good) Object-diagram. But I found it not to be an easy one, and as time went on, as also my project work, I felt that I was really right. So my work proceeded with a parallel design-cum-coding technique, which ofcourse, I agree, is not a good Software Engineering practice. Still, I came out with a handful of *classes (not in C++ , rather, in Object-Oriented terminology)*. I'll try to put that thing here, in somewhat modified form.

The main <u>*Class*</u>es, with their interactions are :



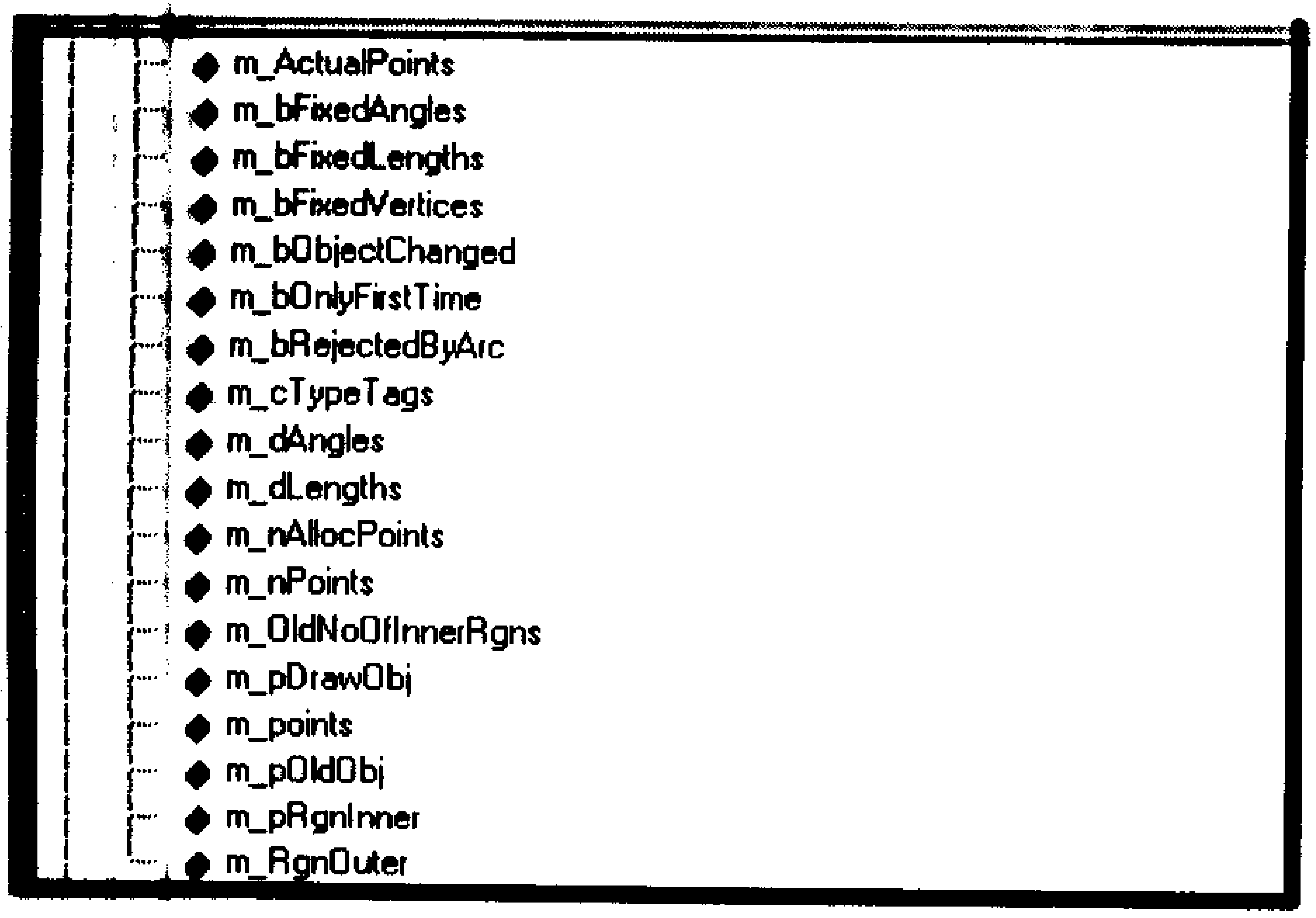
Cview::OnDraw() called when Document changes
The User makes changes to the Shape by either moving handle or by changing property of the Segments

The class "CDrawObj" is the base class for all sort of drawing objects. From it, the classes "CDrawRect" and "CDrawPoly" are derived. The CDrawPoly class is mainly the class for creating "Shapes". This class has got some specialty in that it consists of both line-edges and arc-edges. Below, I show the member-functions and data-members of this class :



```
[illegible]
    AddPoint()
    CDrawPoly(const CRect & position)
    CDrawPoly()
    ~CDrawPoly()
    ChangeToCircle()
    Clone()
    CreateAndTestThisRgn()
    CreateArcRgn()
    CreateThisRgn()
    DeleteEdge()
    Draw()
    DrawEditEdge()
    DrawFixedVertices()
    DrawMarker()
    FindLocusNextTo()
    FindLocusPrevTo()
    FixedAngle()
    FixedLeft()
    FixedRight()
    GetHandle()
    GetHandleCount()
    GetHandleCursor()
    InsertNewVertex()
    Intersects()
    IsCrossIntersecting()
    IsCrossIntersecting(CPolygon & myPolygon, int nHandle, CPoint point)
    IsSelfIntersecting(CPolygon myPolygon, int nHandle, CPoint point)
    IsSelfIntersecting()
    MoveHandleTo()
    MoveTo()
    OnEditCircleProperties()
    OnEditProperties()
    RecalcBounds()
    Serialize()
    SimulatePseudoPolygon(CPolygon & myPolygon, int nHandle, CPoint point)
    SimulatePseudoPolygon(int nHandle, CPoint oldPoint)
```

contd...

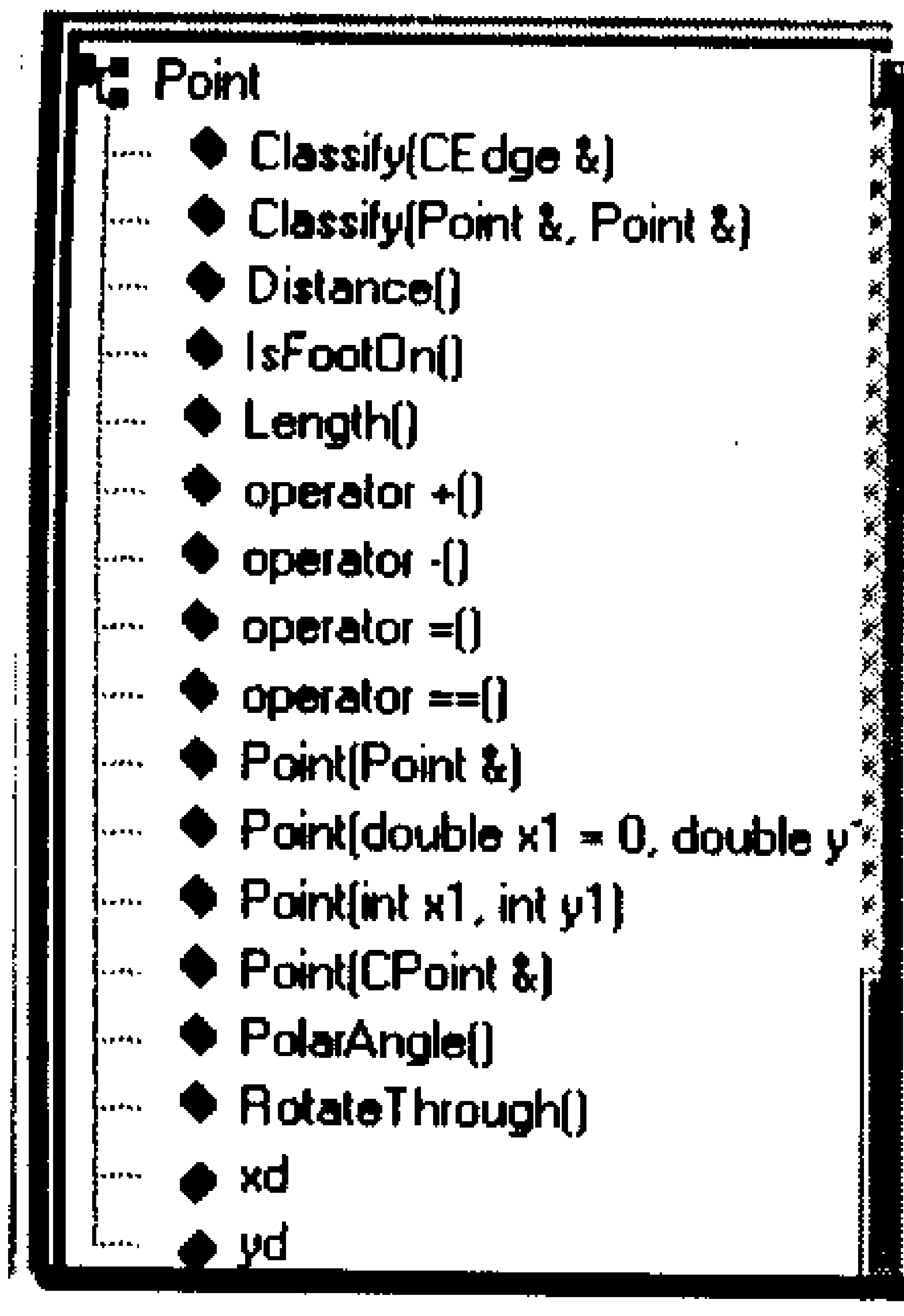


```
◆ m_ActualPoints
◆ m_bFixedAngles
◆ m_bFixedLengths
◆ m_bFixedVertices
◆ m_bObjectChanged
◆ m_bOnlyFirstTime
◆ m_bRejectedByArc
◆ m_cTypeTags
◆ m_dAngles
◆ m_dLengths
◆ m_nAllocPoints
◆ m_nPoints
◆ m_OldNoOfInnerRgns
◆ m_pDrawObj
◆ m_points
◆ m_pOldObj
◆ m_pRgnInner
◆ m_RgnOuter
```

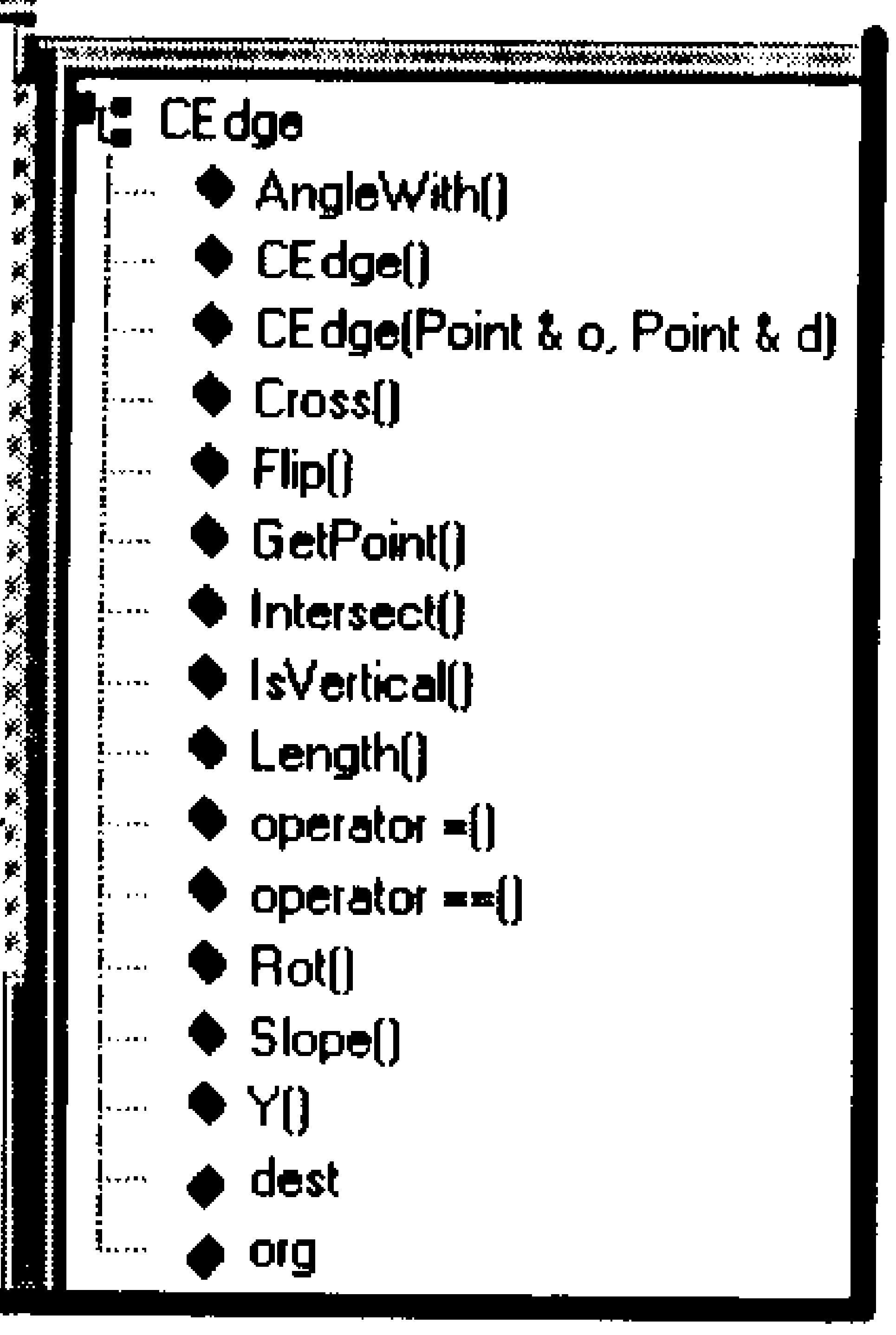
Among the other main classes, the most useful in manipulating two-dimensional geometric shapes, are :

- CPolygon
- CNode
- CVertex
- Point
- CEdge

The "Point" & "CEdge" classes are very frequently used, as and when there is need to change or check properties of the segments of a contour. The member functions of these classes are extensively called to implement the member functions of the class CDrawPoly. Even when dealing with arcs, they proved to be very useful. These classes, I think, constitute the heart of the software. I would like to mention the data-members and member functions of these two classes, in this context. I suppose, the function names are themselves explanatory.



Point
- ◆ Classify(CEdge &)
- ◆ Classify(Point &, Point &)
- ◆ Distance()
- ◆ IsFootOn()
- ◆ Length()
- ◆ operator +()
- ◆ operator -()
- ◆ operator =()
- ◆ operator ==()
- ◆ Point(Point &)
- ◆ Point(double x1 = 0, double y
- ◆ Point(int x1, int y1)
- ◆ Point(CPoint &)
- ◆ PolarAngle()
- ◆ RotateThrough()
- ◆ xd
- ◆ yd



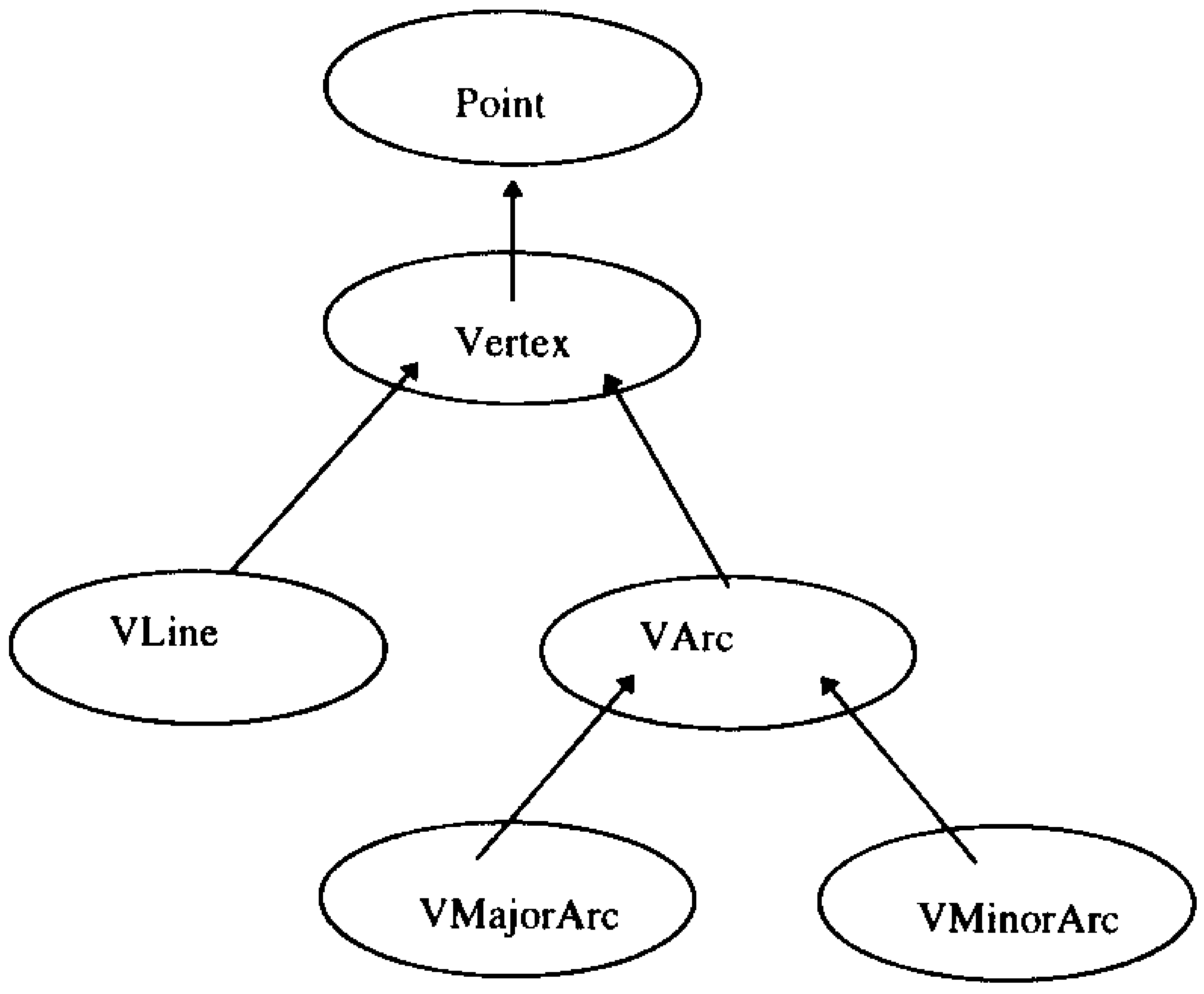
CEdge
- ◆ AngleWith()
- ◆ CEdge()
- ◆ CEdge(Point & o, Point & d)
- ◆ Cross()
- ◆ Flip()
- ◆ GetPoint()
- ◆ Intersect()
- ◆ IsVertical()
- ◆ Length()
- ◆ operator =()
- ◆ operator ==()
- ◆ Rot()
- ◆ Slope()
- ◆ Y()
- ◆ dest
- ◆ org

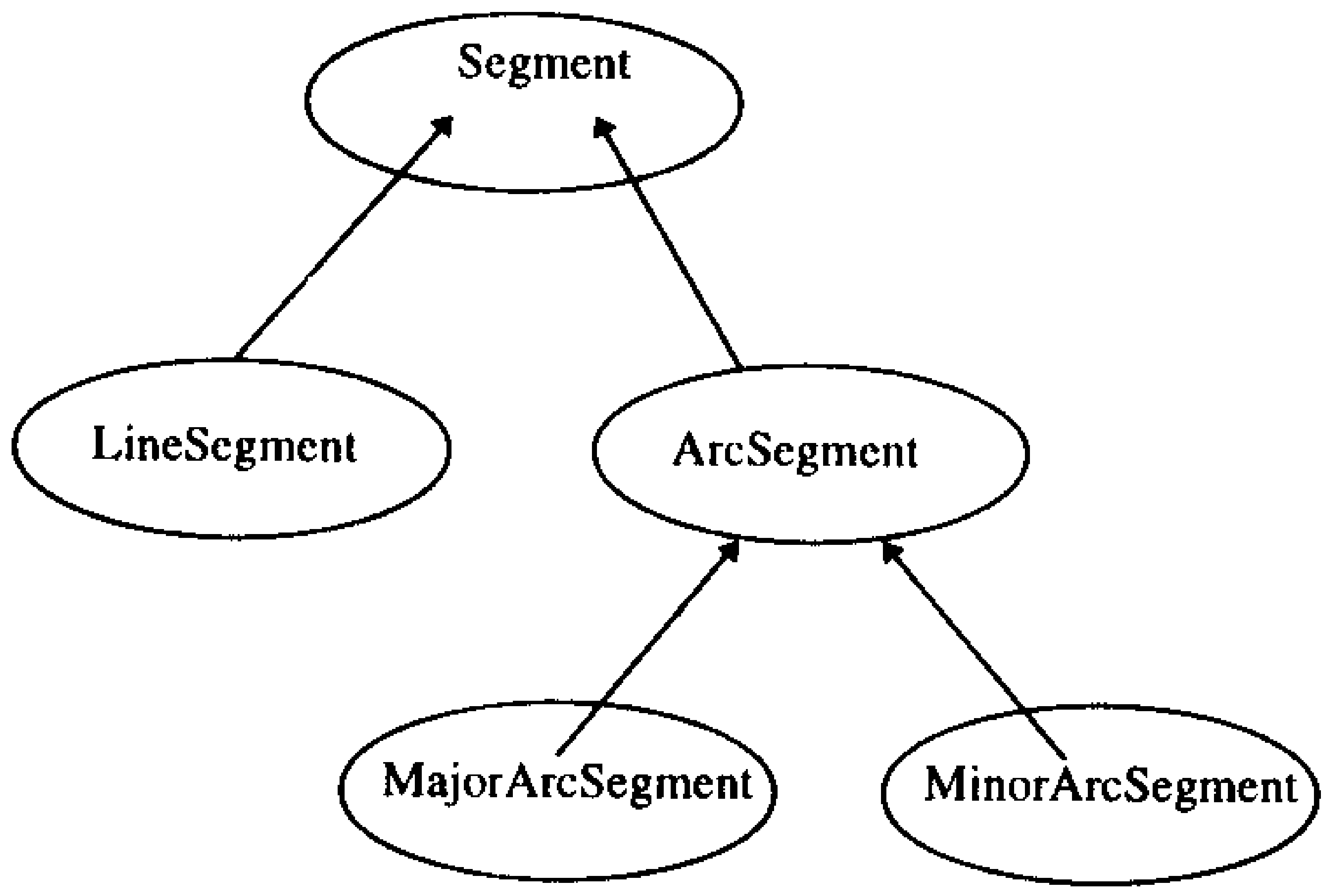
For checking whether any contour (which is basically a CDrawPoly object) overlaps with any other one, I took help of the standard "CRgn" class of MFC. Still, as the contour is not simply a standard polygon, it took some pain to construct the region corresponding to the contour, especially corresponding to the arc-region.

A Better Design :

After undergoing this project work, at this point it seemed to us that there could have been a better object diagram through which the same functionality would have been achieved. We now proceed to give description about that one.

The proposed class-hierarchy is :





The generalized-polygon class "*GPolygon*", can be considered as an aggregate of "*Vertex*".

The attributes and operations defined on these classes are, somewhat as described below :

class ***Point***	::	{ A typical 2-D point }
		// This class can be derived from standard
		// MFC CPoint class & added with
		// capability to store "*float*" coordinates.
class ***Vertex***	::	m_index : integer;
		// Stores the unique index
		// of this *Vertex* inside the
		// GPolygon object.
		IsFixedLength();
		// Is the next edge is of fixed
		// length ?
		IsFixedAngle();
		// Is the next edge has fixed
		// angle wrt. previous one ?
class ***VLine***	::	// attributes & operations specific for
		// line-segments.
class ***VArc***	::	// attributes & operations specific for arcs.
		//eg. m_center : Point;
		// Note : This is an ***abstract class.***
class ***VMajorArc***	::	// extra attributes (if any) &
		// ***VArc*** operations overridden, for arcs of
		// length $\geq$ half-perimeter.
class ***VMinorArc***	::	// extra attributes (if any) &
		// ***VArc*** operations overridden, for arcs of
		// length < half-perimeter.
class ***Segment***	::	m_StartPoint , m_EndPoint : Point;
		// Stores the two endpoints.
		// Note : This is an ***abstract class.***

class ***LineSegment*** :: // extra attributes (if any) for lines &
// ***Segment*** operations overridden.

class ***ArcSegment*** :: // extra attributes (if any) for arcs &
// ***Segment*** operations overridden.
// Note : This is an ***abstract class.***

class ***MajorArcSegment*** :: // extra attributes (if any) &
// ***ArcSegment*** operations overridden, for
// arcs of length $\geq$ half-perimeter.

class ***MinorArcSegment*** :: // extra attributes (if any) &
// ***ArcSegment*** operations overridden, for
// arcs of length $<$ half-perimeter.

The "***Document***" is basically a list of one or more ***GPolygon*** objects, and of the list, the first one is for the outer contour and rests are for inner ones (if any).

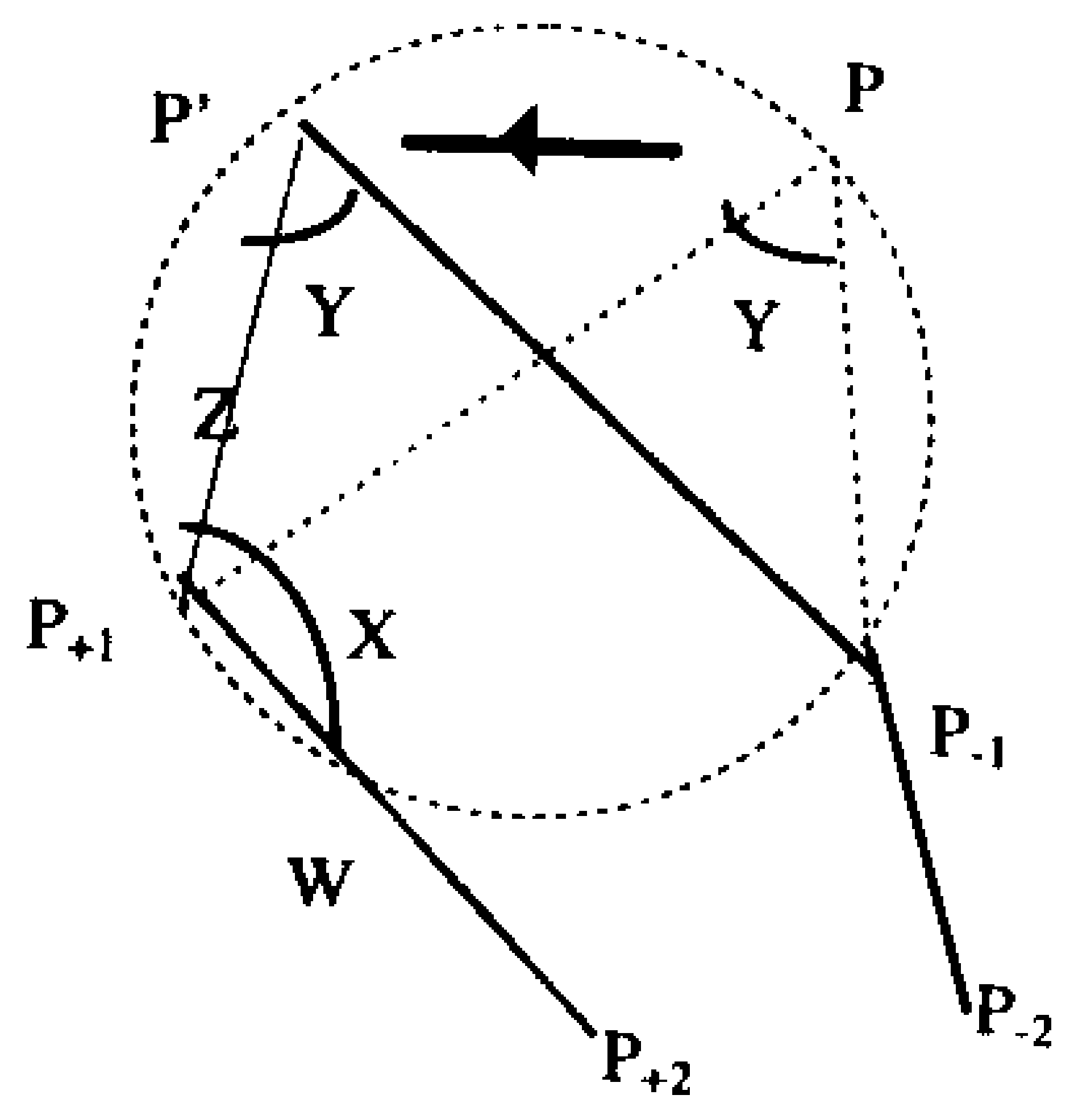
Through the "***View***" class, events as requested by the user, are registered. This class then cross-checks with the ***Document*** to decide, on which ***GPolygon*** object the necessary action is to be taken. Once found the desired object (may be NULL), the corresponding Event-Handler of that object is invoked. This Event-Handler takes care of the desired action on that particular object or on some particular ***Vertex*** or ***Segment*** of that object.

Algorithms :

Though algorithms like, finding out length of a line segment, length of an arc, vector angle of a line with another one, distance of a point from a line segment, classifying a point w.r.t. a line segment, classifying a line segment w.r.t. another one, etc. sound so much trivial, it took some efficient manipulation with the data structure. It would be rather wise to skip the details for these algorithms here.

The only algorithm that would need some discussion, is about simulation of the move of a vertex. Once the user clicks-n-drags a *handle*, the software had to find out the new configuration (i.e. the new positions of other handles) , keeping intact the old relationships.

The algorithm proceeds as follows :



1. Suppose point P is tried to be moved to point P'. We first try to check if at this new position of P, whether there is need to move both of P_{-1} and P_{+1} . If that is the case then we don't allow user to move that point. For the above example, as $P_{-1}P$ doesn't bear constant angle with $P_{-2}P_{-1}$, the effect of moving P to P' is not transferred in backward direction.

2.Now If we find that effect of moving, is in no way transferred to any of the directions, and the constraint is maintained between the edges $P_{-1}P$ and PP_{+1} then we can simply accept that new point.

3.If effect is transferred in only one direction, (in our example, in forward direction, and it can happen only in case of violation of the restricted angle between $P'P_{+1}$ and $P_{+1}P_{+2}$) then allow the move only in cases like :

- X is fixed and only W is fixed, out of both W & Z.
- X is fixed and only Z is fixed, out of both W & Z.

This can be put in a simple form as :

Allow move if $(X \wedge (W \oplus Z))$ is TRUE.

Evaluation Of The Method :

In this topic we will try to explain the problems faced during the course of the software development. First of all, ***Windows Programming*** was like an ocean to me. So the initial couple of months were spent just to learn how to swim across. Then came the idea of ***Document/View architecture***. In Parallel, I was carrying out the algorithm portions. As time passed by, I gradually felt confident to dive into the work of ***Object Identification & Interactions between them***.

After starting implementation, it was found that life was not that much easy. Thanks to ***MFC*** and ***Microsoft Developer Studio*** and their visual tools like ***App Wizard, Class Wizard, Resource Editor*** etc.

Manipulating with regions was a bit troublesome job and it was overcame [illegible] with the use of static CRgn object members inside class CDrawPoly.

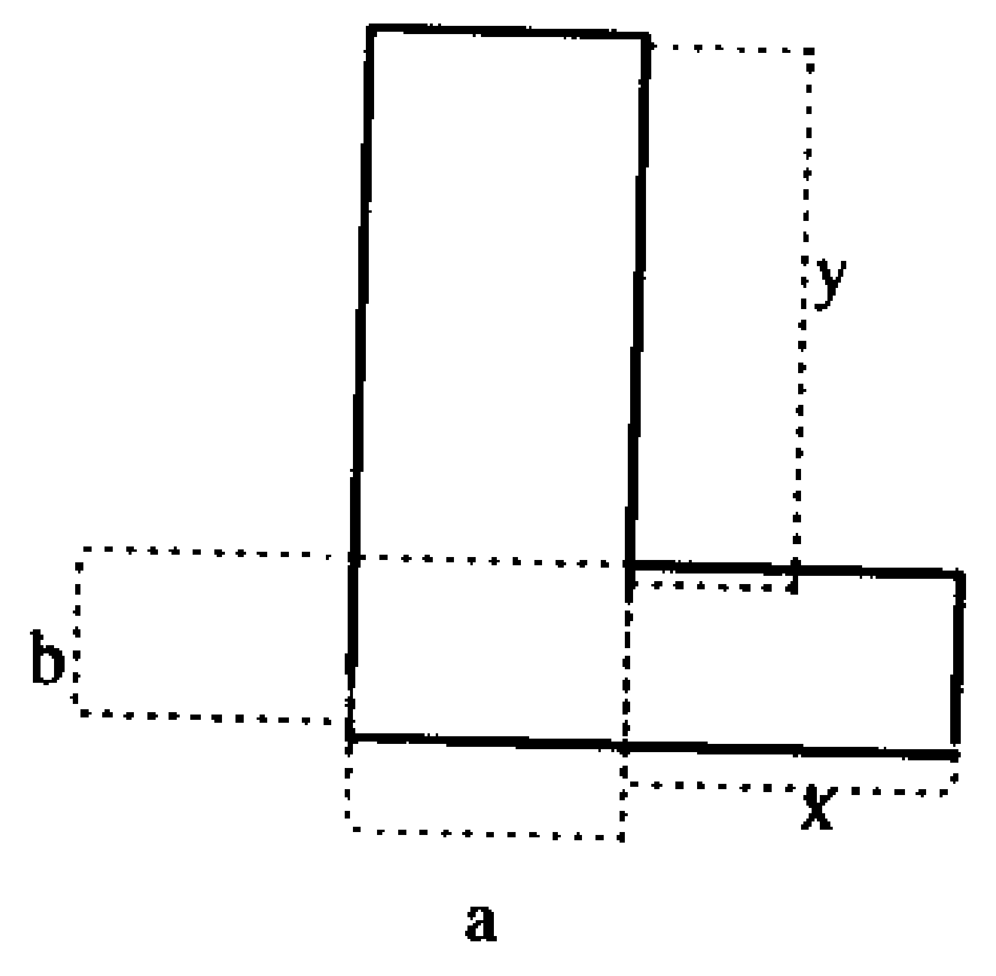
It also was a hard time while drawing [illegible] different segments (especially arcs) of a contour, in the CDrawPoly::Draw() function.

Discussion:

The software finds its uses in the design of parametric shapes. Once designed the figure can be fed to some automated cutting machine to cut out the portions as designed, from a metal/rubber sheet, to get the desired shape.

Parametric shapes are sort of shapes those can be parameterized, i.e. the shape can fully be described by a fixed no of parameters. Only just varying the parameters one can get different instances of the shape. For example, consider an 'L' :

It can fully be represented by 4 parameters a,b,x,y. Only just varying these parameters we get different 'L' shapes, but all are 'L's.



Thus if we can parameterize any shape, then we can create different instances of that shape in future, only by adjusting the parameters. It is also of convenience in that we can store the whole shape in a very condensed format.

Scope Of Further Improvement :

One can extend this dissertation work for 3-dimensional cases. And that is a more real life case, rather than the 2-dimensional one.

References :

1. *Laszlo, Computational Geometry in C++*
2. *Peter Norton, Windows Programming using MFC*
3. *Rambaugh & Others , Object-Oriented Design*